National College of Ireland

Configuration Manual

MSc Research Project
FinTech

Mohammed Aslam Nalakath Assi
Student ID: x22207708

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Mohammed Aslam Nalakath Assi
Student ID:	x22207708
Programme:	FinTech
Year:	2025
Module:	MSc Research Project
Supervisor:	Faithful Onwuegbuche
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Mohammed Aslam Nalakath Assi
Date:	10th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	■
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	■
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	■

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mohammed Aslam Nalakath Assi
x22207708

1 Introduction

The present configuration manual offers a full technical guide of reproducing the machine learning experiments and outcomes presented in the final project *Improving Residential Energy Efficiency Using Machine Learning*. It is targeted at researchers, students, and practitioners that would like to replicate the findings of the paper with the aid of the offered Google Colab notebook, dataset, and execution procedure. The paper outlines the software used, the preparation of data-sets, the preprocessing pipeline, the training and evaluation protocols, and the explainability methods used in this study.

After using the described configuration parameters, snippets of the code, and reproducibility requirements, an independent user should be able to run the whole pipeline and receive the same results as described in the final submission. Suggested execution sequence, run time settings and insertion points of plot are also pointed out in the manual as an aid to clarity and repeatability when new users should rely on them.

2 Software Environment and Dependencies

2.1 Programming Language and Environment

- **Programming Language:** Python 3.10
- **Environment:** Google Colab — a cloud-hosted Jupyter-like environment running on Linux servers.

Python 3.10 is chosen to be compatible with the latest machine learning libraries and Google Colab skips setting up the type of environment and provides cloud computing processing capabilities.

array

2.2 Main Libraries and Packages

Data Handling: `pandas`, `numpy`

Exploratory Data Analysis: `ydata_profiling`

Visualisation: `matplotlib`, `seaborn`, `math`

Preprocessing: `StandardScaler`, `OneHotEncoder`

Machine Learning: `RandomForestClassifier`, `DecisionTreeClassifier`, `LogisticRegression`, `RandomForestRegressor`, `DecisionTreeRegressor`, `LinearRegression`

Evaluation Metrics: `classification_report`, `confusion_matrix`, `mean_squared_error`, `mean_absolute_er` `r2_score`

Explainability: `shap`

The libraries chosen cover all stages of the project end-to-end — data ingestion and preprocessing, model training, evaluation and explanation. The flexibility of the experimentation cannot be overlooked as experimental reproducibility is a given with this type of modular choice.

3 Dataset Configuration and Preprocessing

3.1 Dataset Source

The dataset used in this project is stored in Google Drive at the following path:

```
/content/drive/MyDrive/Practicum/baseline_metadata_and_annual_results.csv
```

It contains building metadata and annual energy performance results for a range of residential properties.

3.2 Initial Cleaning and Transformation

The preprocessing strategy included the following steps:

- Removed columns with more than 80% missing values.
- Classified remaining features into numeric, categorical, and target variables.
- Applied median imputation for missing numeric values.
- Applied most frequent value imputation for missing categorical values.
- Standardised numeric features.
- One-hot encoded categorical variables for machine learning compatibility.

Implementation code for these steps is provided in Section 5.

3.3 Target Variables

- **Classification target:** `efficiency_label` (categories: Efficient, Moderate, Inefficient)
- **Regression target:** `energy_per_sqft` (continuous numeric value)

Cleaning and transformation procedure serves to make the dataset free of superfluous missing data, consistent formats, and ready to use in effective model training. Through scaling and encoding, the project will ensure that there exist both numerical and categorical features that are relevant in the learning process.

4 Experimental Environment Setup

4.1 Runtime Configuration

- Runtime type: Python 3
- Hardware accelerator: None (CPU)
- Session memory: ∼12 GB
- Random seed: `random_state=42` to ensure reproducibility across runs

4.2 Folder Structure

```
Practicum/
|-- baseline_metadata_and_annual_results.csv
|-- Model_Demo.ipynb
|-- Outputs/
| |-- confusion_matrix_rf.png
| |-- shap_summary_rf.png
\-- Reports/
    |-- Final_Report.pdf
    \-- Config_Manual.pdf
```

4.3 Drive Integration

The dataset and supporting files are stored in Google Drive. In Google Colab, the drive was mounted using the following command (see Section 5 for implementation): content/drive/MyDrive/Practicum/

It will be necessary to define a consistent runtime environment and folder structure to collaborate in the project execution. Results can be reproduced by having a fixed random seed, and the workflow is able to read and write to the persistent storage of Google Drive by mounting it, rather than having to re-upload data sets.

5 Key Implementation Code Snippets

5.1 Dataset Loading and Preprocessing

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/Practicum/baseline_metadata_and_annual_results.csv
    ')

# Identify numeric and categorical columns
numeric_cols = X_train.select_dtypes(include=['float64','int64']).columns.tolist()
categorical_cols = X_train.select_dtypes(include=['object','category']).columns.tolist()
```

```
# Scaling numeric features
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import numpy as np
scaler = StandardScaler()
X_train_num = scaler.fit_transform(X_train[numeric_cols])
X_test_num = scaler.transform(X_test[numeric_cols])

# Encoding categorical features
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_train_cat = encoder.fit_transform(X_train[categorical_cols])
X_test_cat = encoder.transform(X_test[categorical_cols])

# Combine numeric and categorical arrays
X_train_final = np.hstack([X_train_num, X_train_cat])
X_test_final = np.hstack([X_test_num, X_test_cat])
```

In this code, Google Drive is mounted where the dataset is loaded and feature matrix is prepared by scaling and one-hot encoding which ensures that both numeric and categorical variables are formated correctly to be used in machine learning models.

5.2 Classification Models

5.2.1 Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

model_rf = RandomForestClassifier(random_state=42, n_jobs=-1)
model_rf.fit(X_train_final, y_train)
y_pred_rf = model_rf.predict(X_test_final)

print(classification_report(y_test, y_pred_rf, target_names=['Efficient','Moderate','
    Inefficient']))
print(confusion_matrix(y_test, y_pred_rf))
```

5.2.2 Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_final, y_train)
y_pred_dt = dt_classifier.predict(X_test_final)
```

5.2.3 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression(max_iter=500, solver='lbfgs', multi_class='multinomial',
    n_jobs=-1)
logreg_model.fit(X_train_final, y_train)
y_pred_logreg = logreg_model.predict(X_test_final)
```

5.3 Regression Models

5.3.1 Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_final, df['energy_per_sqft'], test_size=0.2, random_state=42
)

reg_model = RandomForestRegressor(random_state=42, n_jobs=-1)
reg_model.fit(X_train_reg, y_train_reg)
y_pred_reg = reg_model.predict(X_test_reg)
```

5.3.2 Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
dt_regressor = DecisionTreeRegressor(random_state=42)
dt_regressor.fit(X_train_reg, y_train_reg)
y_pred_dtr = dt_regressor.predict(X_test_reg)
```

5.3.3 Linear Regression

```
from sklearn.linear_model import LinearRegression
linreg_model = LinearRegression()
linreg_model.fit(X_train_reg, y_train_reg)
y_pred_lr = linreg_model.predict(X_test_reg)
```

5.4 Explainability with SHAP

5.4.1 SHAP for Decision Tree Regressor

```
import shap, numpy as np, pandas as pd
sample_idx = np.random.choice(len(X_test_reg), 100, replace=False)
X_small_dtr_df = pd.DataFrame(X_test_reg[sample_idx], columns=numeric_cols + encoder.
    get_feature_names_out(categorical_cols).tolist())
explainer_dtr = shap.TreeExplainer(dt_regressor)
shap_values_dtr = explainer_dtr.shap_values(X_small_dtr_df)
shap.summary_plot(shap_values_dtr, X_small_dtr_df, max_display=20)
```

5.4.2 SHAP for Random Forest Regressor

```
rf_small = RandomForestRegressor(n_estimators=50, max_depth=6, random_state=42)
rf_small.fit(X_train_reg[:10000], y_train_reg[:10000])

sample_idx = np.random.choice(len(X_test_reg), 50, replace=False)
```

```
X_small_rf_df = pd.DataFrame(X_test_reg[sample_idx], columns=numeric_cols + encoder.
    get_feature_names_out(categorical_cols).tolist())

explainer_rf = shap.Explainer(rf_small, X_small_rf_df)
shap_values_rf = explainer_rf(X_small_rf_df)
shap.summary_plot(shap_values_rf, X_small_rf_df, max_display=20)
```

SHAP explains individual model predictions by giving a value to each feature that shows its contribution to the prediction making it easy to predict what each feature is contributing to the model output and enhancing the transparency and trustworthiness of model output predictions.

6 Model Training and Evaluation

6.1 Classification Results

Table 1: Classification Accuracy and F1-Macro Scores

Model	**Accuracy**	**F1-Macro**
Random Forest Classifier	0.87	0.88
Decision Tree Classifier	0.87	0.87
Logistic Regression	0.81	0.82

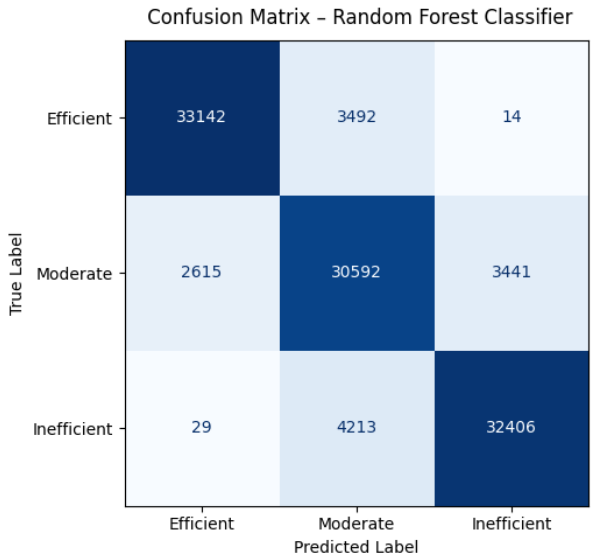

Figure 1: Confusion Matrix — Random Forest Classifier

Random Forest Classifier performs best in F1-Macro which depicts an evenly balanced performance on all efficiency rating. Logistic Regression is not promising implying that the non- linear models fit better on this dataset.

6.2 Regression Results

Table 2: Regression Performance Metrics

Model	R^2 Score	MAE	MSE
Random Forest Regressor	0.9426	1.4937	6.0318
Decision Tree Regressor	0.8885	1.9828	11.7127
Linear Regression	0.6902	4.0270	32.5272

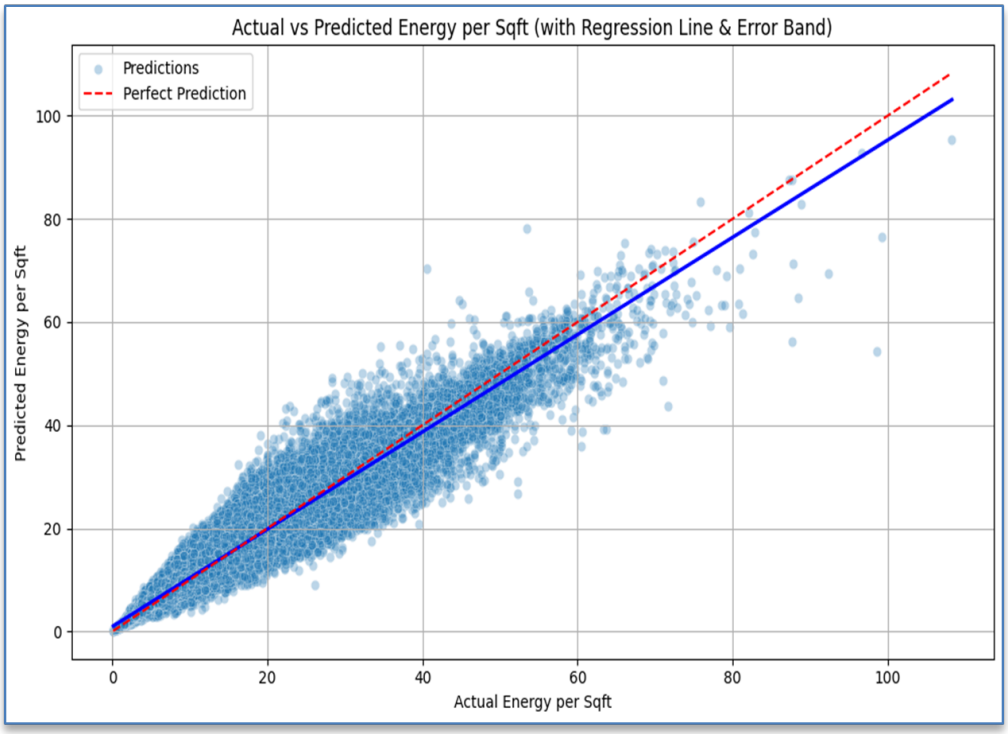

Figure 2: Actual vs. Predicted — Random Forest Regressor

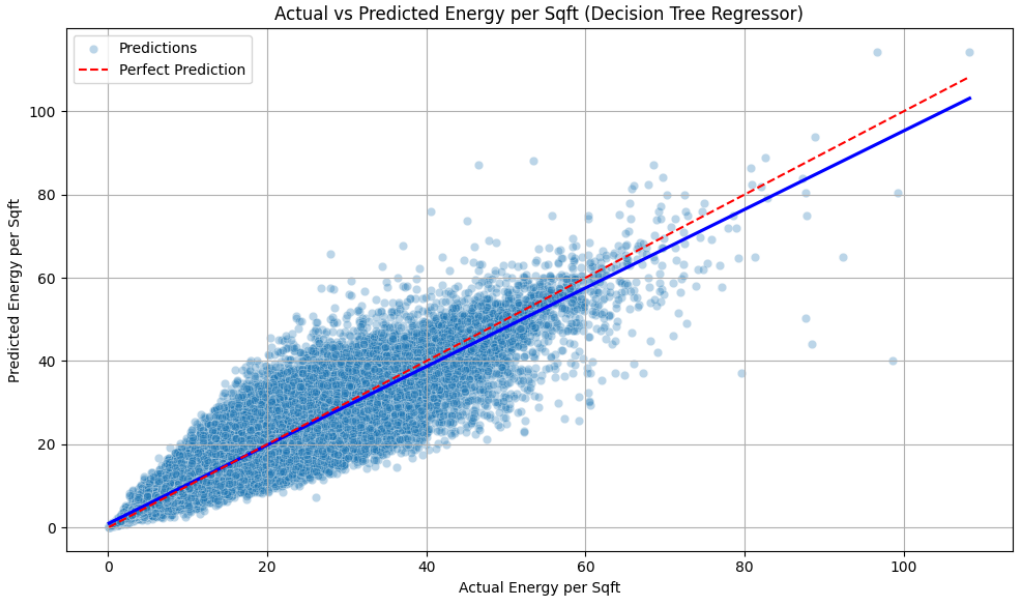

Figure 3: Actual vs. Predicted — Decision Tree Regressor

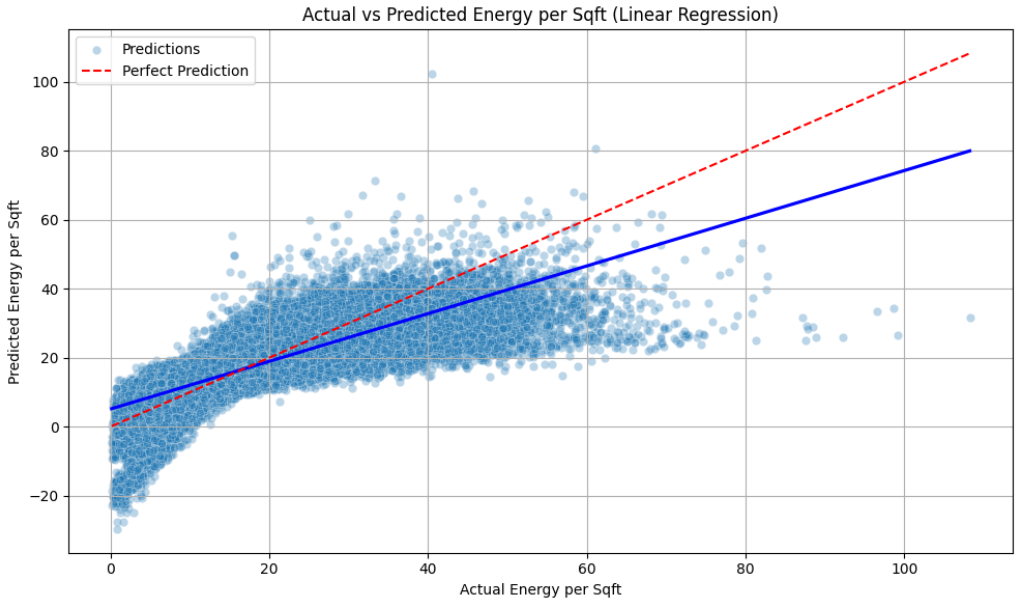

Figure 4: Actual vs. Predicted — Linear Regression

Random Forest Regressor is the best compared to the other regression models as it has the highest R^2 and lowest errors. Performance gap is an indication of some high non-linear relationships in the data which is well explained by the tree-based methods.

6.3 Best Models and Explainability

Best Classification Model: Random Forest Classifier (Accuracy: 0.87, F1-Macro: 0.88) **Best Regression Model:** Random Forest Regressor (R^2: 0.9426, MAE: 1.4937, MSE: 6.0318)

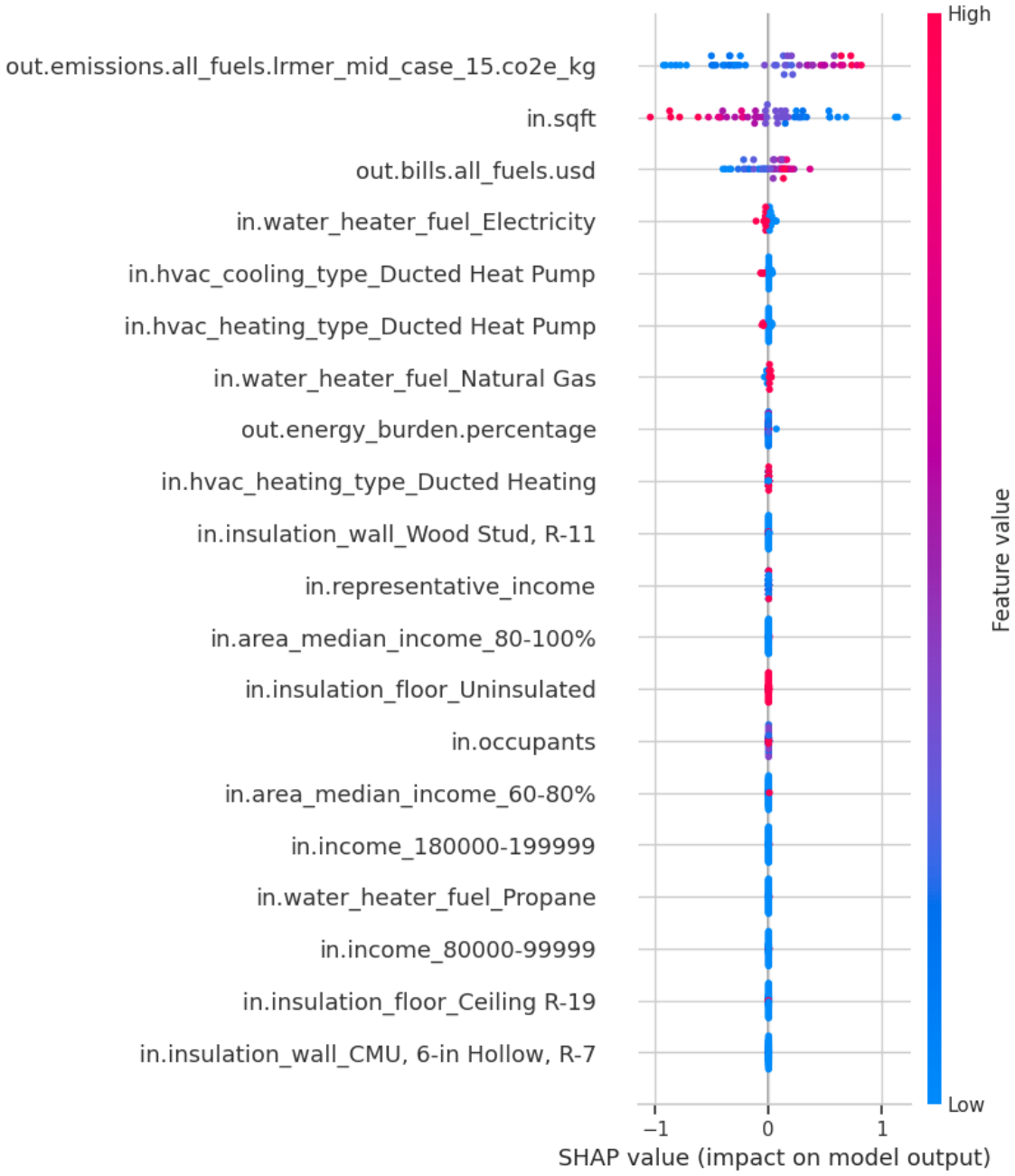

Figure 5: SHAP Summary Plot — Random Forest Regressor

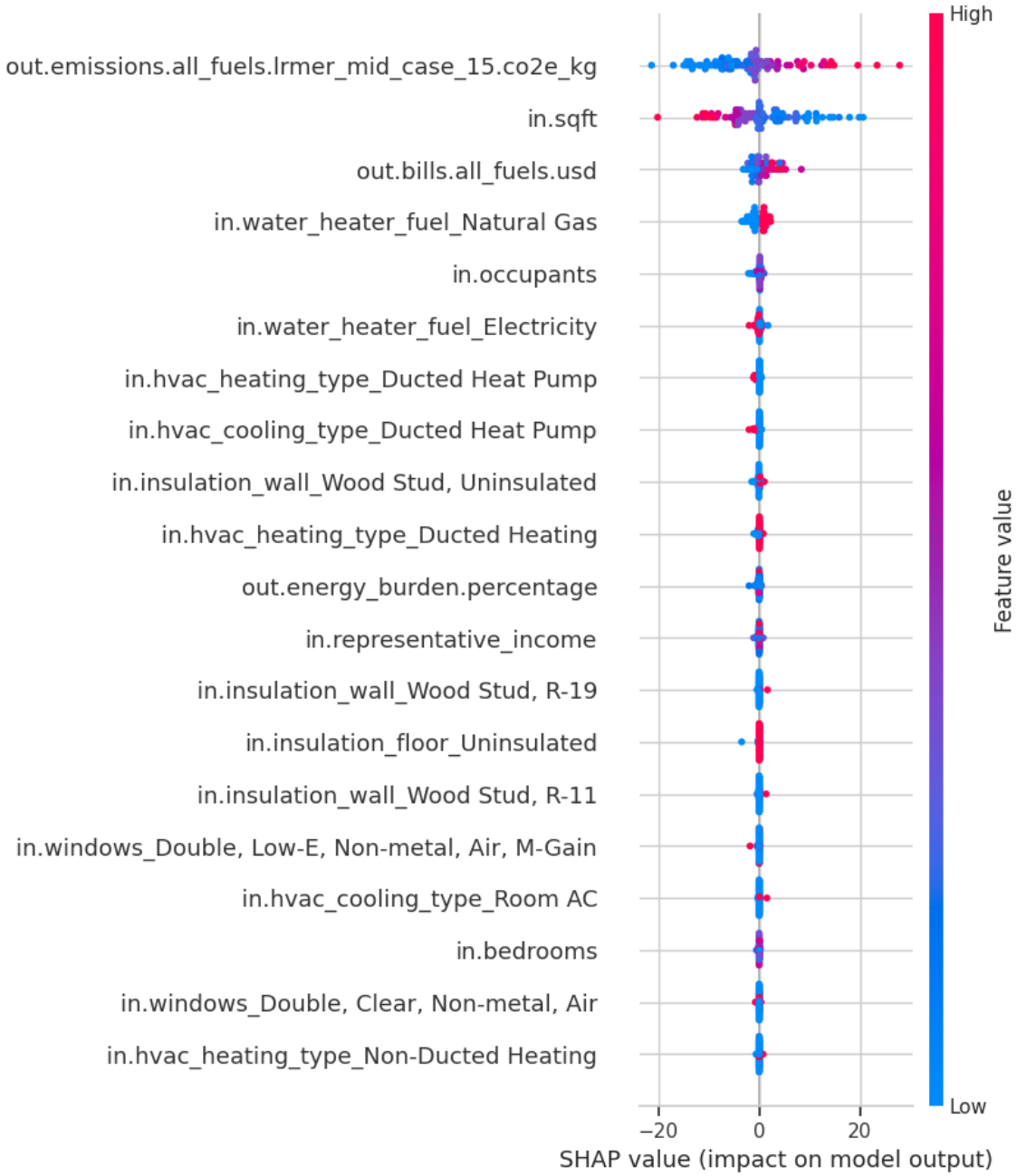

Figure 6: SHAP Summary Plot — Decision Tree Regressor Regressor

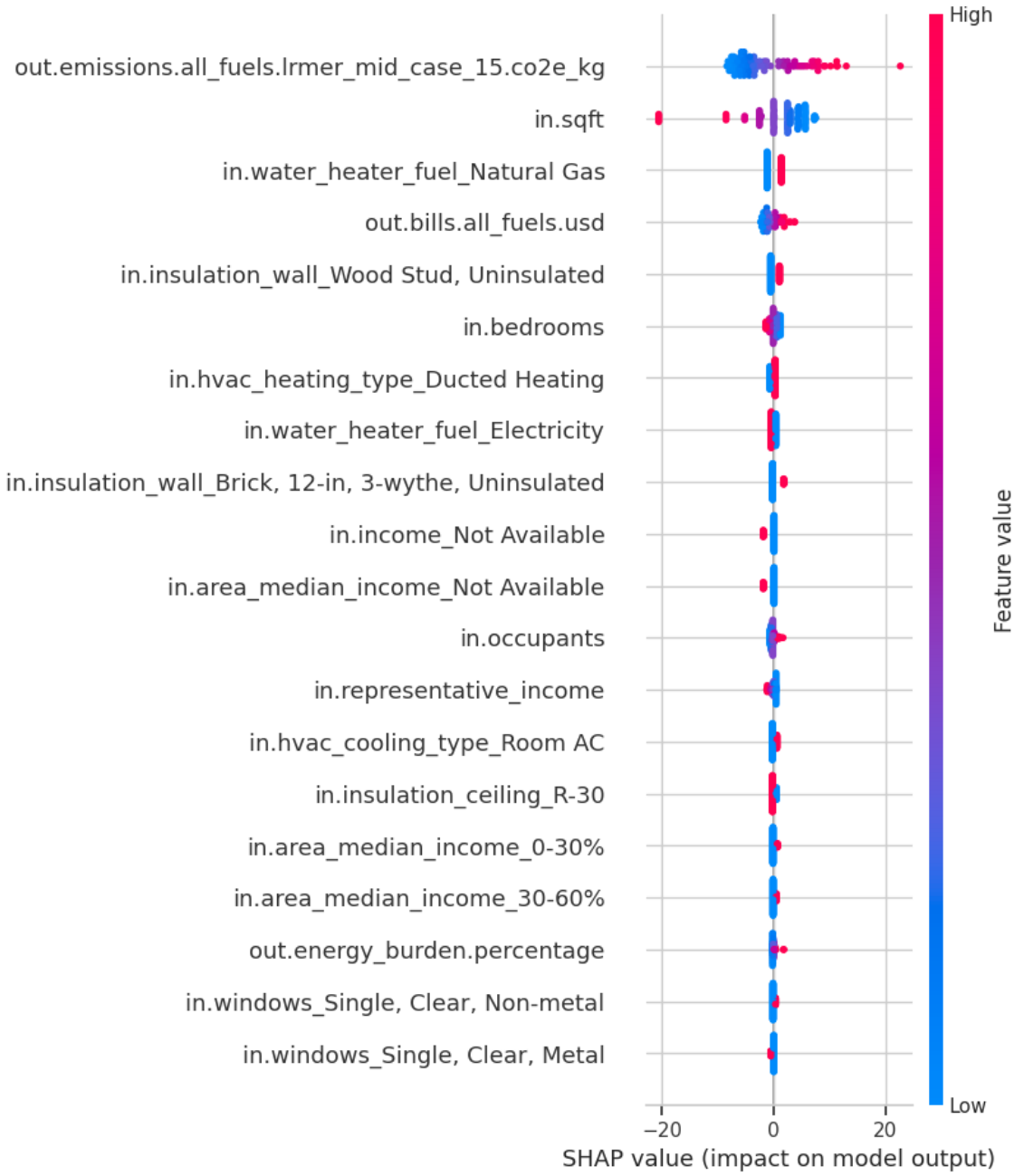

Figure 7: SHAP Summary Plot — Linear Regression

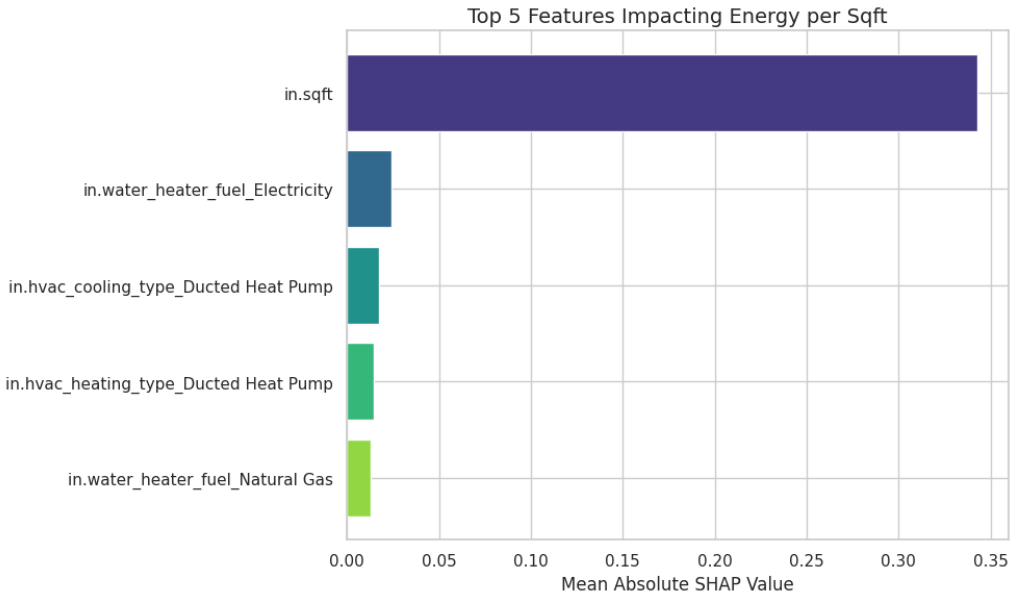

Figure 8: Top Features by Mean |SHAP| — Random Forest Regressor

As the SHAP analysis shows, it is possible to establish the most powerful features of any prediction, which is useful information about the relationship between building characteristics and energy efficiency determination drawn by the models. This justifies explainability and model dependability.

7 Reproducibility Guidelines

7.1 Steps to Reproduce the Results

n order to achieve the same results follow in the following order:

1. **Open the Google Colab Notebook:** Upload `Model_Demo.ipynb` to Google Colab.
2. **Set Runtime Configuration:** Select `Python 3` as the runtime type and choose `None (CPU)` as the hardware accelerator.
3. **Mount Google Drive:** Connect your Google Drive to Colab so the dataset is accessible. Use the code in Section 5.
4. **Load Dataset:** Load the dataset from the path `/content/drive/MyDrive/Practicum/baseline_metadata_and_annual_results.csv`.
5. **Preprocess the Data:** Apply the scaling and encoding procedures as described in Section 5.
6. **Train Models:** Run the classification and regression models sequentially, starting with Random Forest.
7. **Generate Evaluation Metrics and Plots:** Execute the code to produce confusion matrices, actual vs. predicted plots, and SHAP visualisations.

8. **Interpret Results:** Compare outputs to the values reported in Section 6 to confirm replication accuracy.

7.2 Additional Notes for Reproducibility

- Ensure all package versions match those listed in Section 2.
- Execute the notebook cells in sequential order from top to bottom without skipping.
- If running in a different environment (e.g., local Jupyter Notebook), ensure file paths are updated accordingly.

It standardizes data set loading, preprocessing, and model training which are essential to the reproducibility of the published results. Sequential runs, deterministic random number seeds, and synchronized libraries remove all sources of variability caused by differences in the environment or by differences in procedure.

8 Conclusion

This configuration manual has given a list of all components and a methodical presentation of information to create the same experimental conditions as the ones implemented in the project *Improving Residential Energy Efficiency Using Machine Learning*. It discussed the software environment, preparation of dataset, pre-processing logic, experimental setup, code of implementation, assessment of the model, and the reproducibility practices.

Upon the described procedures, a separate user will be able to duplicate the results as stated in the final report. The transparency of the models is also increased due to the inclusion of explainability methods, like SHAP, that gives stakeholders the ability to trust and understand the predictions.

Although this manual is strong in encouraging strict replication, it also offers ground on which future experimentations can be based. The users can go further with data analysis by discussing other feature engineering techniques, the improvement path of hyper parameters, or other machine learning models to widen and deepen the analysis.

This final part is a restatement affirming that the manual is a replication guide yet it is an innovation kick off point. The systematic process is scientifically reproducible but provides space to explore and adapt it to different sets of data or research objectives.